Electronic Version
Stylesheet Version v1.1.1

Description

Extensible Decimal Identification System for Ordered Nodes

Background of Invention

Field of Invention

[0001] The present invention relates generally to the field of database management systems. More specifically, the present invention is related to a decimal identification system for ordered nodes.

Discussion of Prior Art

[0002] A tree structure comprising nodes is a type of data structure in which each element is attached to one or more elements directly beneath it. The connections among elements in a tree structure are called branches. Trees are often called inverted trees because they are normally drawn with the root at the top. Inverted trees are the data structures used to represent hierarchical file structures. In this case, the leaves are files and the other elements

above the leaves are directories.

[0003] Tree structures have been used in prior art data processing systems to organize data. But, such prior art fails to provide for a node identification system for ordered nodes wherein adding or deleting a child node (or a subtree of nodes) from a hierarchical structure of nodes still maintains the order and relationships between the parent, child, and sibling nodes.

[0004] Node identification solutions based upon assigning preorder traversal and/or postorder traversal numbers can only provide an ordering solution as such solutions cannot be used to identify sibling relationships between nodes.

[0005] Figure 1 illustrates a node identification solution wherein a concatenation of values are assigned. The root node is assigned an ID of 0. The root node's first child is assigned an ID composed of the parent's ID (0) and a 0. The root node's second child is assigned an ID composed of the parent's ID (0) and a 1. This process is repeated until all nodes are assigned IDs. This system establishes a total ordering for the nodes, i.e., 0 < 0.0 < 0.0.0 < 0.0.1 < 0.0.2 < 0.0.2.0 < 0.0.2.1 < 0.1 < ... 0.2.2. In addition, the system identifies parent, child and sibling relationships. That is, 0.1 is the parent of 0.1.0 and 0.1.1 because

0.1 is the prefix in both children, and that 0.1.0 and 0.1.1 are siblings because they have the same prefix.

[0006] The problem with this system, shown in figure 1, is that changes to the hierarchy would require the re-numbering of IDs to maintain the order of the nodes. For example, as in Figure 2a, adding a node B would require the re-numbering of the node with an ID of 0.2.2, and adding a node C would require the re-numbering of nodes with IDs 0.0.0, 0.0.1, and 0.0.2, including re-numbering any children of such nodes. Adding a node A, however, would not require re-numbering because the new node can be assigned an ID of 0.0.2.2.

[0007] Figure 2b illustrates the modified hierarchical structure of nodes after the insertion of node A, B, and C. The filled nodes (0.0.1, 0.0.2, 0.0.3, 0.0.3.0, 0.0.3.1, and 0.2.3) represent nodes whose values have changed due to the insertion of nodes A, B, or C. The double circle nodes (0.0.0, 0.0.3.2, and 0.2.2) represent nodes that are newly inserted. Hence, it can be seen that the insertion of a new node (or, similarly, the deletion of an existing node) causes a change in the ID values associated with other nodes in a hierarchical structure of nodes.

[0008] The following references provide a general teaching in or-

dering nodes, but they fail to provide for a solution wherein existing ID values can remain the same even after changes (such as insertion or deletion of nodes) are made in a hierarchical structure of nodes.

[0009] The patent to Jordan, Jr. (5,063,502) provides for an apparatus and method for controlling concurrent process access of infrastructures comprising tree structures of complex object nodes. The apparatus associated with each complex object node records an accumulate count of each type of lock applied by concurrently running computer processes against each infrastructure complex node.

[0010] The patent to Kothuri et al. (6,505,205) provides for a system and method for indexing and storing multi-dimensional or multi-attribute data. Data items are recursively sorted in a selected dimension (e.g., the dimension having the greatest variance) and divided until each subdivision fits into a leaf node having a specified fanout. Intermediate nodes and a root node are constructed to complete the index. Each node of the index is stored in a database as a separate object or record and may include a node identifier of the unique, an identifier of a parent and/or a sibling node and an entry for each child of the

node, which may be data items or other nodes. Each record entry for a child includes an associated bounding area encompassing descendant data items. Another database table or module may store information about the index, such as the dimensionality of the data, the index fanout and an identifier of a root of the index.

[0011] The patent application publication to Keating (2002/0052895) discloses a system and method for generalizing the content in a formatted document, wherein the system and method permit a group with multiple elements to be processed rapidly even if the number of elements in the group changes over time.

[0012] Whatever the precise merits, features, and advantages of the above cited references, none of them achieves or fulfills the purposes of the present invention.

SUMMARY OF INVENTION

[0013] The present invention provides for an extensible identification system for nodes in a hierarchy, wherein each node is assigned a concatenation of decimal based values. The identification value uniquely identifies the node, provides an order for the node, and identifies its parent, child, and sibling relationships with other nodes. Also, the IDs assigned can be encoded to be byte comparable. Further-

more, the ID's assigned to nodes need not be modified when changes (adding/deleting a child node or a subtree of nodes) are made in the hierarchy. Additionally, in the event of such a change, the order and relationships between the parent, child, and sibling nodes are retained.

[0014] The present invention provides for a robust method for updating a computer-stored hierarchical structure of nodes via a node identification technique, wherein the method comprises the steps of: (a) receiving an instruction to insert a new node at an insertion point in a computer-stored hierarchical structure; (b) identifying one of, or a combination of the following: a left node ID value closest to the left of the insertion point or a closest right node ID value closest to the right of the insertion point; (c) calculating a new ID value via any of the following steps: concatenating the left node ID value with one or more high key values and a positive value, decreasing last digit of the right node ID value, increasing last digit of left node ID value, decreasing last digit of the right node ID value and concatenating a positive value, or concatenating the left node ID value with one or more zeros and a positive value, wherein the calculated value is greater than ID values of nodes to the left of the insertion point and less

than ID values of nodes to the right of the insertion point; and (d) updating the computer-stored hierarchical structure by inserting the new node in the hierarchy and associating the new node with the calculated ID value. As a result of such an implementation, the order, node ID values, and relationships between parent, child, and siblings in the hierarchical structure of nodes remain unchanged with the insertion of new nodes.

[0015] The present invention provides a way for assigning IDs to nodes in a hierarchy and provides many advantages, some of which include: (a) the IDs provide a way of ordering nodes in a hierarchy; (b) the IDs describe a node's parent, child, and sibling relationships; (c) the IDs can be encoded such that they are byte comparable; (d) the IDs can be assigned to newly inserted nodes, anywhere in the hierarchy, and still maintain these properties; and (e) the IDs, once assigned, do not have to change even with changes to the hierarchy.

BRIEF DESCRIPTION OF DRAWINGS

[0016] Figure 1 illustrates a hierarchical node structure representative of the prior art.

[0017] Figures 2a and 2b collectively illustrate the concept of renumbering when nodes A, B, and C are added to an ex-

isting hierarchical node structure.

[0018] Figures 3 through 8 collectively illustrate various embodiments of the present invention's technique to avoid re-numbering of existing node IDs.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0019] While this invention is illustrated and described in a preferred embodiment, the invention may be implemented in many different configurations. There is depicted in the drawings, and will herein be described in detail, a preferred embodiment of the invention, with the understanding that the present disclosure is to be considered as an exemplification of the principles of the invention and the associated functional specifications for its construction and is not intended to limit the invention to the embodiment illustrated. Those skilled in the art will envision many other possible variations within the scope of the present invention.

[0020] The present invention provides for a method for inserting nodes in a hierarchically ordered node structure without modifying existing node IDs. Figure 3 illustrates one embodiment of the present invention for avoiding the re-numbering of IDs. In this embodiment, 1 (instead of 0 as in Figures 1 and 2a-b) starts the count on every level. The

method is triggered when an instruction is received to insert a new node at an insertion point in said computer-stored hierarchical structure. If a node is inserted in between two siblings, the digits of the left sibling are concatenated with a high key value 'x' and a 1. In one instance, to insert a node in between left node ID 1.1 and right node ID 1.2, digits of the left sibling, 1.1, are concatenated with x and 1 to give 1.1.x.1. The prefix 1.1.x guarantees that any node in the subtree of the left sibling 1.1 is lower in value. The positive 1 after the high key 'x' starts off a count. To insert another node in between left node ID 1.1.x.1 and right node ID 1.2, the last digit of the left node ID is increased, giving the new node an ID value of 1.1.x.2. To insert another node in between left node ID 1.1 and right node ID 1.1.x.1, the last digit of the right node ID is decreased, giving the new node an ID value of 1.1.x.-1.

[0021] Analogous to the high key value 'x', a low key value '0' is used to extend the count in the opposite direction. For example, to insert a node in between 1.1.x.(-1) and 1.1.x.1, an ID value of 1.1.x.0.1 is used. Negative numbers are used when counting in descending order to avoid using the low key '0'. The level of a node can be deter-

mined from the ID by subtracting the number of x's multiplied by 2, and the number of 0's from the total number of digits. For example, 1.1.x.1 belongs to level 2.

[0022] Figure 4 illustrates another example of the above-mentioned embodiment wherein nodes are inserted between a node with an ID value of 4.4 and a node with an ID value of 4.5. As in figure 3, to insert a node between left node ID 4.4 and right node ID 4.5, digits of the left sibling, 4.4, are concatenated with x and 1 to give 4.4.x.1. Similarly, to insert a node in between 4.4.x.(-1) and 4.4.x.1, an ID value of 4.4.x.0.1 is used. To insert another node in between left node ID 4.4.x.1 and right node ID 4.5, the last digit of the left node ID is increased, giving the new node an ID value of 4.4.x.2. To insert another node in between left node ID 4.4 and right node ID 4.4.x.1, the last digit of the right node ID is decreased, giving the new node an ID value of 4.4.x.-1. Tables 1 and 2 show an example of how the IDs of Figure 4 are encoded. It should be noted that the encoding shown is byte comparable.

Binary Code	Symbol
0001 0111 1111	-9
0001 1000	-8
0001 1110	-2
0001 1111	-1
0010	0
0011	1
0100	2
0101	3
0110	4
0111	5
1000 0000	$5 + 2^0$
1001 1111	$5 + 2^5$
1010 1111 1111	$5 + 2^5 + 2^8$
1011 0111 1111 1111	$..+ 2^8 + 2^{11}$
1111	x
0000	Reserved
0001	Negative Sign

Table 1

Symbol	Binary Code
4.4	0110 0110
4.4.x.(-1)	0110 0110 1111 0001 1111
4.4.x.0.1	0110 0110 1111 0010 0011
4.4.x.1	0110 0110 1111 0011
4.4.x.1.x.1	0110 0110 1111 0011 1111 0011
4.4.x.2	0110 0110 1111 0100
4.5	0110 0111

Table 2

[0023] Figure 5 illustrates a variation of the technique of figures 3-4. The variation involves avoiding using 0's and instead using x's to extend the series. For example, 4.4.x.(-1).x.1 is used to follow 4.4.x.(-1), instead of 4.4.x.0.1. This leaves both x and 0 free to be used as positive infinity and negative infinity, which can be used as special purpose high/low key values as shown in Tables 3 and 4.

Binary Code	Symbol
0001 0111 1111	-9
0001 1000	-8
0001 1110	-2
0001 1111	-1
0010	0
0011	1
0100	2
0101	3
0110	4
0111	5
1000 0000	$5 + 2^0$
1001 1111	$5 + 2^5$
1010 1111 1111	$5 + 2^5 + 2^8$
1011 0111 1111 1111	$..+ 2^8 + 2^{11}$
1111	+x Positive Infinity
0000	-x Negative Infinity
0001	Negative Sign

Table 3

Symbol	Binary Code
4.4	0110 0110
4.4.x.(-2)	0110 0110 1111 0001 1110
4.4.x.(-1)	0110 0110 1111 0001 1111
4.4.x.(-1).x.1	0110 0110 1111 0001 1111 1111 0011
4.4.x.1	0110 0110 1111 0011
4.4.x.1.x.1	0110 0110 1111 0011 1111 0011
4.4.x.2	0110 0110 1111 0100
4.5	0110 0110 0111

Table 4

[0024] To compute a number that is higher than any of 4.5's left sibling, or any of its descendant, the last number is decremented and concatenated with x.x resulting in 4.4.x.x This number is higher than any of 4.5's left sibling, or any of the siblings descendant. This property of the encoding (i.e., the fact that a number such as this can be computed) is important when doing a sort. This aspect is show in Figure 6. For example, a sort instruction can be issued to return all ID values between 4.4 and 4.4.x.x for identifying all of 4.5's previous siblings and any of its descendants.

[0025] Same can be said for 4.4.0.1. Children and descendants of 4.4 can be identified by issuing a sort instruction to return all ID values between higher than 4.4.0.1, but not greater than 4.4.x.x.

[0026] Another property about this encoding is that these numbers can be compute to use as keys for searching. For example, one need not have prior knowledge of whether 4.5's previous sibling is 4.3, 4.2, or 4.4 (since during deletes, nodes 4.4, 4.3, etc., could be removed). Such numbers can be computed and used as special keys to find the existence of sibling/parent/children nodes.

[0027] Figure 7 illustrates another embodiment wherein, in lieu

of using a high key value 'x', a gap is introduced in the count between nodes and the gap value is used as the high key value. For example, sibling nodes are assigned 1.1, 1.3, and 1.5, and so on. To insert a node in between left node ID 1.1 and right node ID 1.3, the last digit of the right node ID is decreased and concatenated with 1, giving the newly inserted node an ID value of 1.2.1. The prefix 1.2 guarantees that any node in the subtree of the left sibling 1.1 is lower in value. To insert another node between left node ID 1.2.1 and right node ID 1.3, the last digit of the left node ID is increased, giving the newly added node an ID value of 1.2.3. To insert a node in between nodes with ID values of 1.2.1 and 1.2.3, an ID value of 1.2.2.1 is used. The 0 digit and negative numbers are used in the same way as in Figure 3. To insert a new node between left node ID 1.1 and right node ID 1.2.0.1, the last digit of the right node ID is decreased, giving the newly added node an ID value of 1.2.-1. The level of an ID can be determined by counting the number of odd digits in the ID. For example, the ID 1.2.2.1 belongs to level 2.

[0028] Figure 8 illustrates another example of the above-mentioned embodiment wherein nodes are inserted between a node with an ID value of 7.7 and a node with an

ID value of 7.9. As in Figure 5, to insert a node in between nodes with ID values of 7.7 and 7.9, an ID value of 7.8.1 is used. Similarly, to insert another node after 7.8.1, an ID value of 7.8.3 is used and to insert a node in between nodes with ID values of 7.8.1 and 7.8.3, an ID value of 7.8.2.1 is used. Tables 5 and 6 show an example of how the IDs of Figure 8 are encoded. It should be noted that the encoding shown is byte comparable.

Binary Code	Symbol
0001 0111 1111	-9
0001 1000	-8
0001 1110	-2
0001 1111	-1
0010	0
0011	1
0100	2
0101	3
0110	4
0111	5
1000 0000	$5 + 2^0$
1011 1111	$5 + 2^6$
1101 1111 1111	$5 + 2^{6+}2^9$
1110 1111 1111 1111	$..+ 2^9 + 2^{12}$
0000	Reserved
0001	Negative Sign

Table 5

Symbol	Binary Code
7.7	1000 0001 1000 0001
7.8.(-1)	1000 0001 1000 0010 0001 1111
7.8.0.1	1000 0001 1000 0010 0010 0011
7.8.1	1000 0001 1000 0010 0011
7.8.2.1	1000 0001 1000 0010 0100 0011
7.8.3	1000 0001 1000 0010 0101
7.9	1000 0001 1000 0011

Table 6

[0029] Some of the benefits of the present invention are: (a) nodes in a hierarchy are identifiable by a unique ID; (b) each ID describes whether a node is a parent, a child or a sibling of another; (c) the IDs provide a total ordering; (d)the IDs can be encoded such that they are byte comparable; (e) no matter where nodes are inserted, the newly encoded IDs are still byte comparable and maintains order; and (f) existing IDs in a hierarchy can stay the same even with changes (insertions or deletions) made to the hierarchy.

[0030] Additionally, the present invention provides for an article of manufacture comprising computer readable program code contained within implementing one or more modules updating a computer-stored hierarchical structure of nodes via a node identification technique, wherein such an

update allows for retaining the properties and parent/child relationships of the hierarchical structure without renumbering existing node ID values associated with the hierarchical structure. Furthermore, the present invention includes a computer program code-based product, which is a storage medium having program code stored therein which can be used to instruct a computer to perform any of the methods associated with the present invention. The computer storage medium includes any of, but is not limited to, the following: CD-ROM, DVD, magnetic tape, optical disc, hard drive, floppy disk, ferroelectric memory, flash memory, ferromagnetic memory, optical storage, charge coupled devices, magnetic or optical cards, smart cards, EEPROM, EPROM, RAM, ROM, DRAM, SRAM, SDRAM, or any other appropriate static or dynamic memory or data storage devices.

[0031] Implemented in computer program code based products are software modules for: (a) receiving an instruction to insert a new node at an insertion point in a computer-stored hierarchical structure; (b) identifying one of, or a combination of the following: a left node ID value closest to the left of the insertion point or a closest right node ID value closest to the right of the insertion point; (c) calcu-

lating a new ID value via any of the following steps: concatenating the left node ID value with one or more high key values and a positive value, decreasing last digit of the right node ID value, increasing last digit of left node ID value, decreasing last digit of the right node ID value and concatenating a positive value, or concatenating the left node ID value with one or more zeros and a positive value, the calculated value greater than ID values of nodes to the left of the insertion point and less than ID values of nodes to the right of the insertion point; and (d) updating the computer-stored hierarchical structure by inserting the new node wherein order, node ID values, and relationships between parent, child, and siblings in the hierarchical structure of nodes remain unchanged with the insertion of new node.

CONCLUSION

[0032] A system and method has been shown in the above embodiments for the effective implementation of an extensible decimal identification system for ordered nodes. While various preferred embodiments have been shown and described, it will be understood that there is no intent to limit the invention by such disclosure, but rather, it is intended to cover all modifications falling within the spirit

and scope of the invention, as defined in the appended claims. For example, the present invention should not be limited by software/program, computing environment, or specific computing hardware.

[0033] The above enhancements are implemented in various computing environments. For example, the present invention may be implemented on a conventional IBM PC or equivalent, multi-nodal system (e.g., LAN) or networking system (e.g., Internet, WWW, wireless web). All programming and data related thereto are stored in computer memory, static or dynamic, and may be retrieved by the user in any of: conventional computer storage, display (i.e., CRT) and/or hardcopy (i.e., printed) formats. The programming of the present invention may be implemented by one of skill in the art of database programming.